«The work presented in this book is based on a simple intuition:

if substrate systems like programming languages, object systems, databases or operating systems can be tailored to meet particular application needs as they arise, rather than having to hack around existing deficiencies, application writers are better off.»

Gregor Kiczales and Andreas Pæpcke
Open Implementation and Meta-Object Protocol (MOP)
Separation of Concerns (SoC)
Bibliography

Black- and Gray-Box Approaches.

**Black Box**
- the accesses to the system functionality is limited to the mechanisms provided by the adopted programming language;
- an attempt of using the system functionality can raise an "application mismatch" when a component is used in the wrong way;
- the flexibility is really limited.

**Gray Box**
- open implementation;
- the component behavior can be adapted to our needs;
- we can by-pass the mechanisms provided by the programming language to access the system functionality;
- we can re-class the objects respecting their use and behavior.

Open Implementation & Meta-Object Protocol.
Kinds of Opening.

It is possible to evince three ways of opening up the access to the system details:
- **introspection**, is the system ability of observing the state and the structure of the system itself;
- **intercession**, is the system ability of modifying the behavior and the structure of the system itself;
- **invoke**, is the system ability of applying the system functionality.

Examples of MOP.

Non-typed and interpreted programming languages
- Lisp
  - CLOS (Gregor Kiczales, 1991), ObjVLisp (Pierre Cointe, 1987), ABCL-R (Akinori Yonezawa, 1988);
- SmallTalk
  - NeoClassTalk, MetaClassTalk (Mohammed N. Bouraqadi-Saâdani, 1998);
  - Apertos (Yasuhiro Yokote, 1992), CodaA (Jeff McAffer, 1995).

Typed and interpreted programming languages
- Java
  - java.lang.reflect (Sun, 1995);
  - OpenJava (Michiaki Tatsubori, 1999), Javassist (Shigeru Chiba, 2000), Reflex (Eric Tanter, 2001).

Compiled programming languages
- C/C++
Separation of Concerns (SoC).

Introduction.

**Complete Application**

= **Core Functionality**
  (e.g., banking applications: accounts, clients, operations, ...)
  + **Nonfunctional Concerns**
    (security, persistence, distribution, exception handling, concurrency, ...)

Note that the separation between functional and nonfunctional is not so clear and neat.

Traditionally,
- separation of concerns is at design stage only;
- source code is a mixture of all concerns (functional and nonfunctional)
  - error prone;
  - bad reusability and extensibility.

SOC aims at keeping separate the concerns at implementation stage
- reflection, aspect-oriented programming;

Reflection allows the designer of separating the functional aspects from the nonfunctional ones.

Therefore, we get [*]:
- an augmentation of the functionality reuse;
- an augmentation of the system stability; and
- the functional and nonfunctional aspects can be developed independently.

Walter Hürsch and Cristina Videira Lopes.
Separation of Concerns.
Technical Report NU-CCS-95-03, Northeastern University,
Boston, February 1995.